Basic Plotting with matplotlib

You can show matplotlib figures directly in the notebook by using the `%matplotlib notebook` and `%matplotlib inline` magic commands.

`%matplotlib notebook` provides an interactive environment.

In [1]:
```
%matplotlib notebook
```

In [2]:
```
import matplotlib as mpl
mpl.get_backend()
```

Out[2]:
```
'nbAgg'
```

In [3]:
```
import matplotlib.pyplot as plt
plt.plot?
```

In [36]:
```
# because the default is the line style '-',
# nothing will be shown if we only pass in one point (3,2)
plt.plot(3, 2)
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x7f92e8a8ca20>]
```

In [35]:
```
# we can pass in '.' to plt.plot to indicate that we want
# the point (3,2) to be indicated with a marker '.'
plt.plot(3, 2, '.')
```

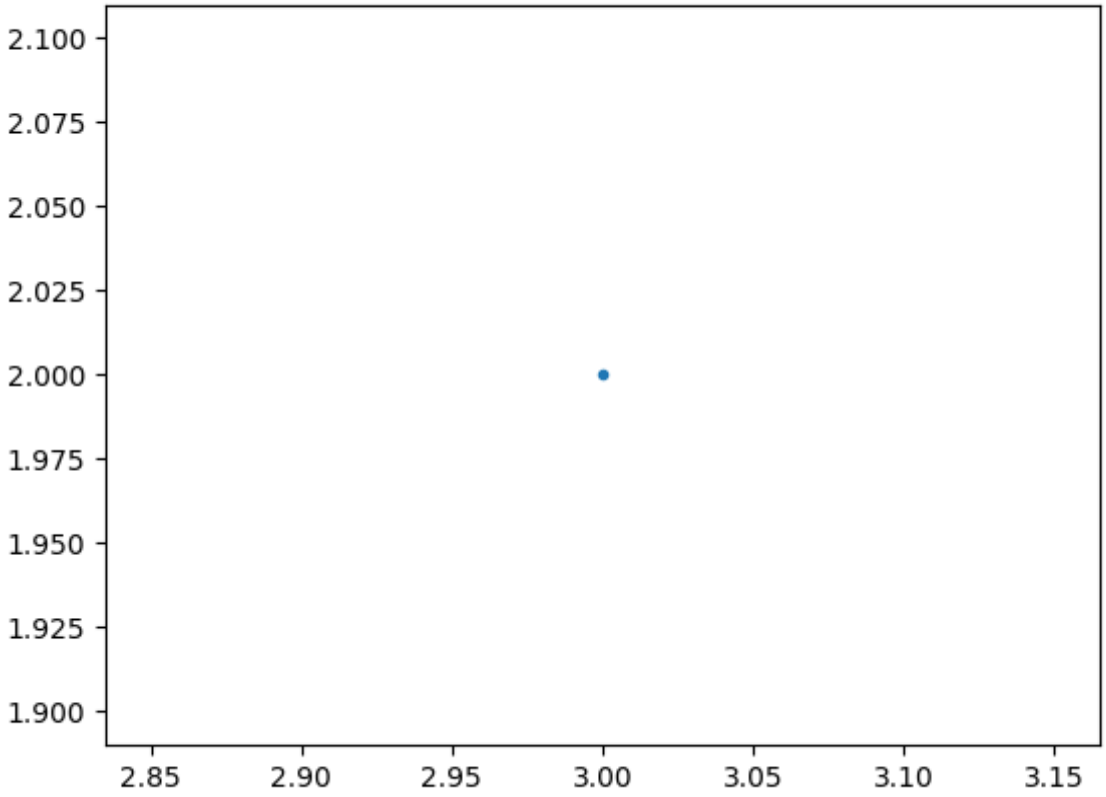
Out[35]:
```
[<matplotlib.lines.Line2D at 0x7f92e89b3550>]
```

Let's see how to make a plot without using the scripting layer.

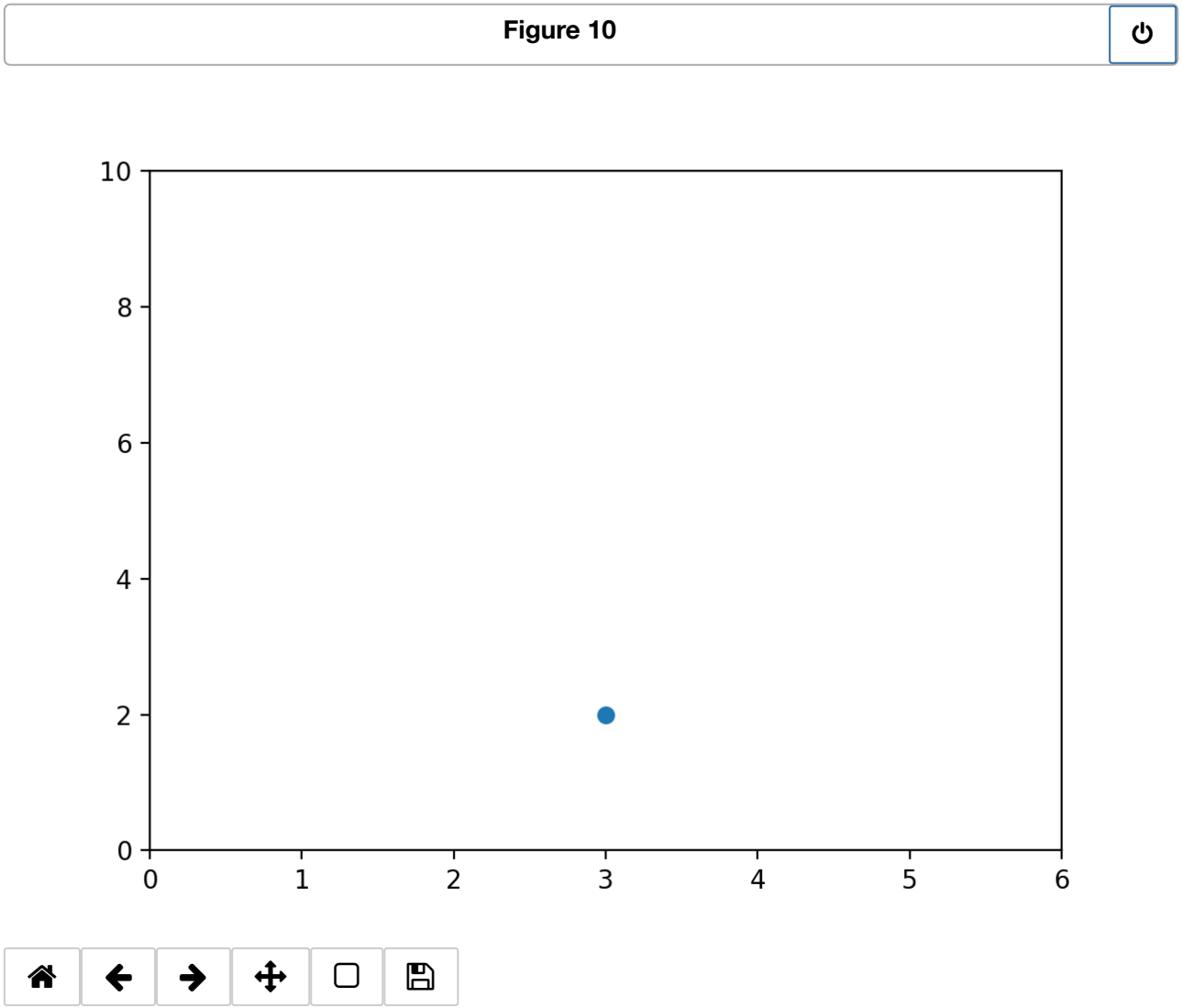
In [6]:
```
# First let's set the backend without using mpl.use() from the scripting layer
from matplotlib.backends.backend_agg import FigureCanvasAgg
from matplotlib.figure import Figure

# create a new figure
fig = Figure()

# associate fig with the backend
canvas = FigureCanvasAgg(fig)

# add a subplot to the fig
ax = fig.add_subplot(111)

# plot the point (3,2)
ax.plot(3, 2, '.')

# save the figure to test.png
# you can see this figure in your Jupyter workspace afterwards by going to
# https://hub.coursera-notebooks.org/
canvas.print_png('test.png')
```

We can use html cell magic to display the image.

In [7]:
```
%%html
<img src='test.png' />
```

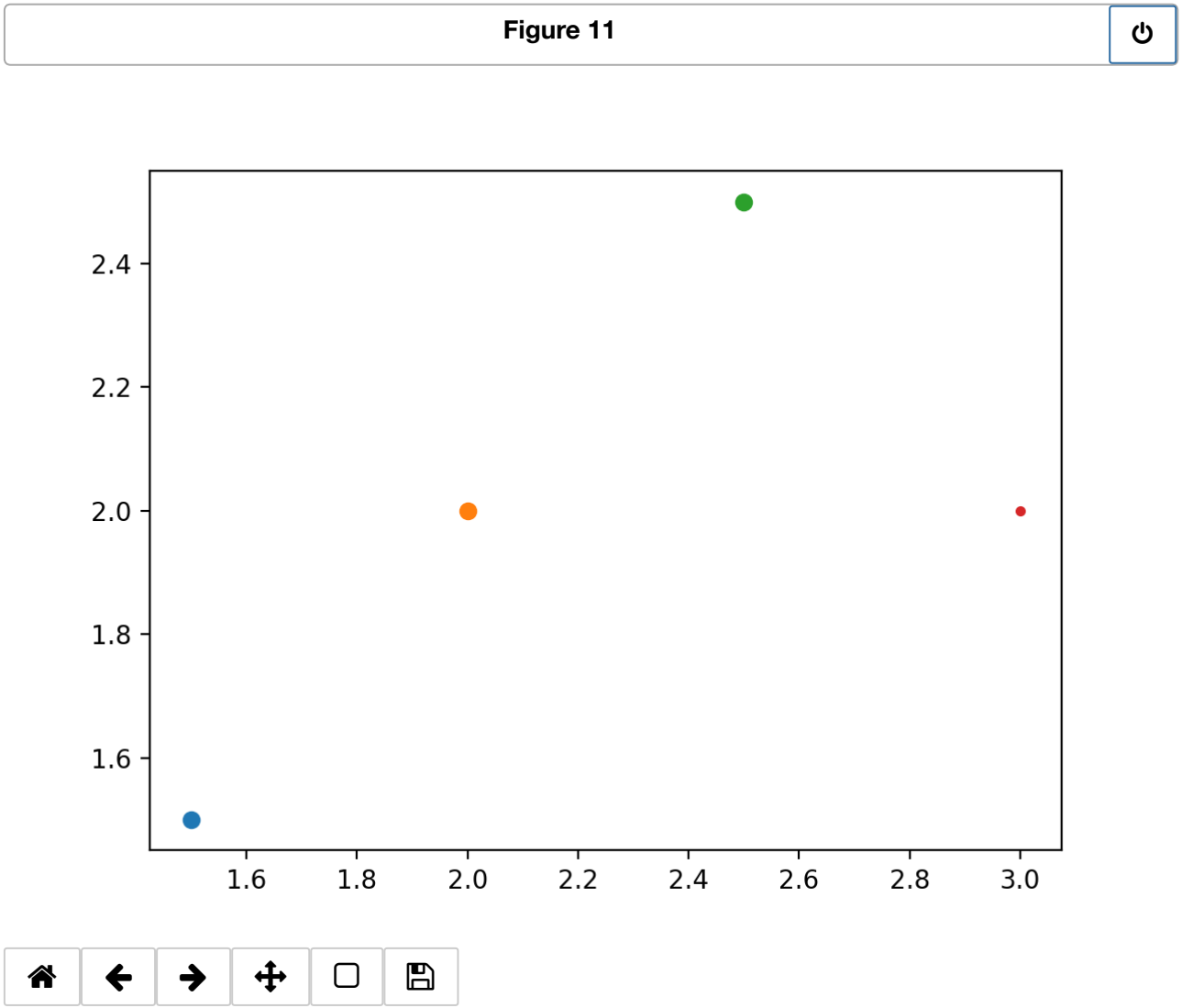


In [33]:
```
# create a new figure
plt.figure()

# plot the point (3,2) using the circle marker
plt.plot(3, 2, 'o')

# get the current axes
ax = plt.gca()

# Set axis properties [xmin, xmax, ymin, ymax]
ax.axis([0,6,0,10])
```




```
Out[33]: [0, 6, 0, 10]
```

In [34]:
```
# create a new figure
plt.figure()

# plot the point (1.5, 1.5) using the circle marker
plt.plot(1.5, 1.5, 'o')
# plot the point (2, 2) using the circle marker
plt.plot(2, 2, 'o')
# plot the point (2.5, 2.5) using the circle marker
plt.plot(2.5, 2.5, 'o')
```




```
Out[34]: [<matplotlib.lines.Line2D at 0x7f92e8963cc0>]
```

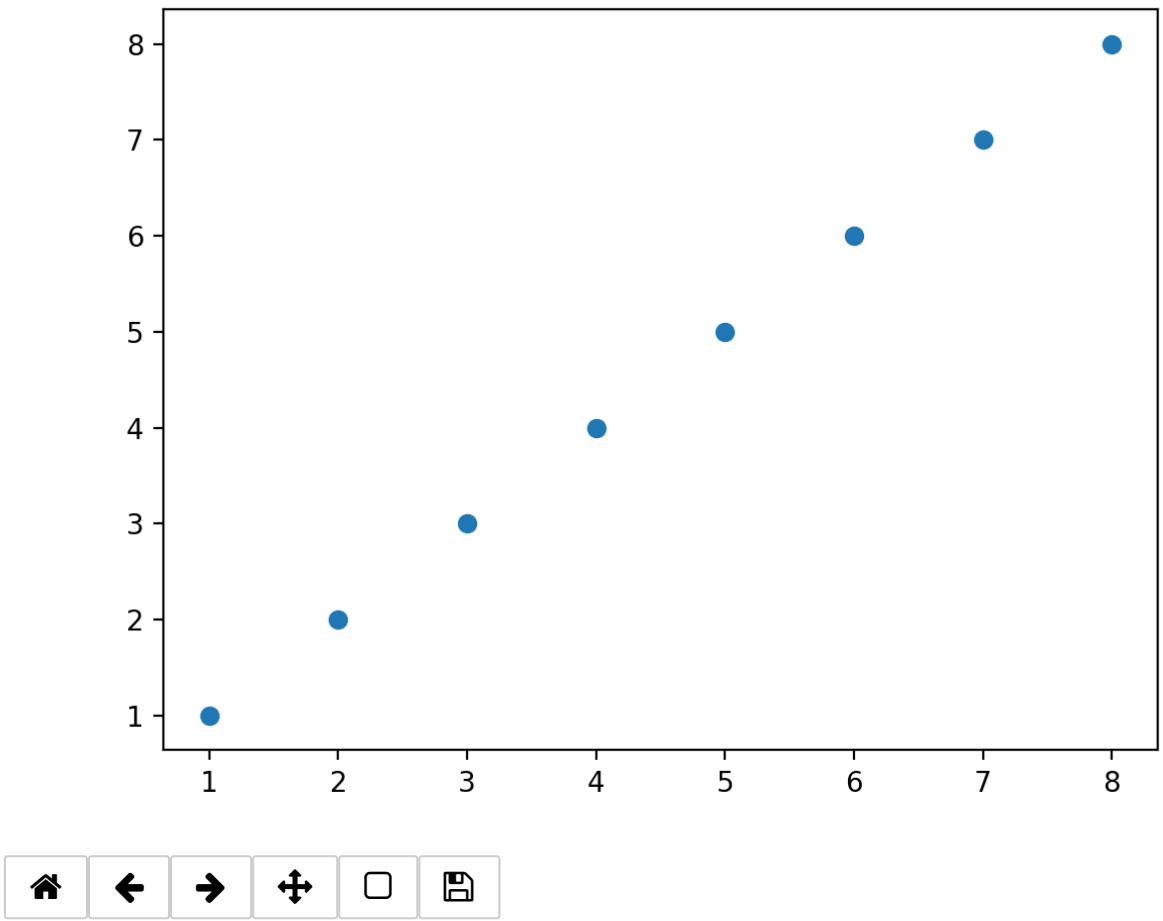
In [10]:
```
# get current axes
ax = plt.gca()
# get all the child objects the axes contains
ax.get_children()
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x7f92eaf6d5c0>,
 <matplotlib.lines.Line2D at 0x7f92ed1f0710>,
 <matplotlib.lines.Line2D at 0x7f92eaf6df98>,
 <matplotlib.spines.Spine at 0x7f92eafac080>,
 <matplotlib.spines.Spine at 0x7f92eafac2b0>,
 <matplotlib.spines.Spine at 0x7f92eafac4a8>,
 <matplotlib.spines.Spine at 0x7f92eafac6a0>,
 <matplotlib.axis.XAxis at 0x7f92eafac860>,
 <matplotlib.axis.YAxis at 0x7f92eafb2d68>,
 <matplotlib.text.Text at 0x7f92eaf495f8>,
 <matplotlib.text.Text at 0x7f92eaf49048>,
 <matplotlib.text.Text at 0x7f92eaf49160>,
 <matplotlib.patches.Rectangle at 0x7f92eaf49080>]
```

Scatterplots

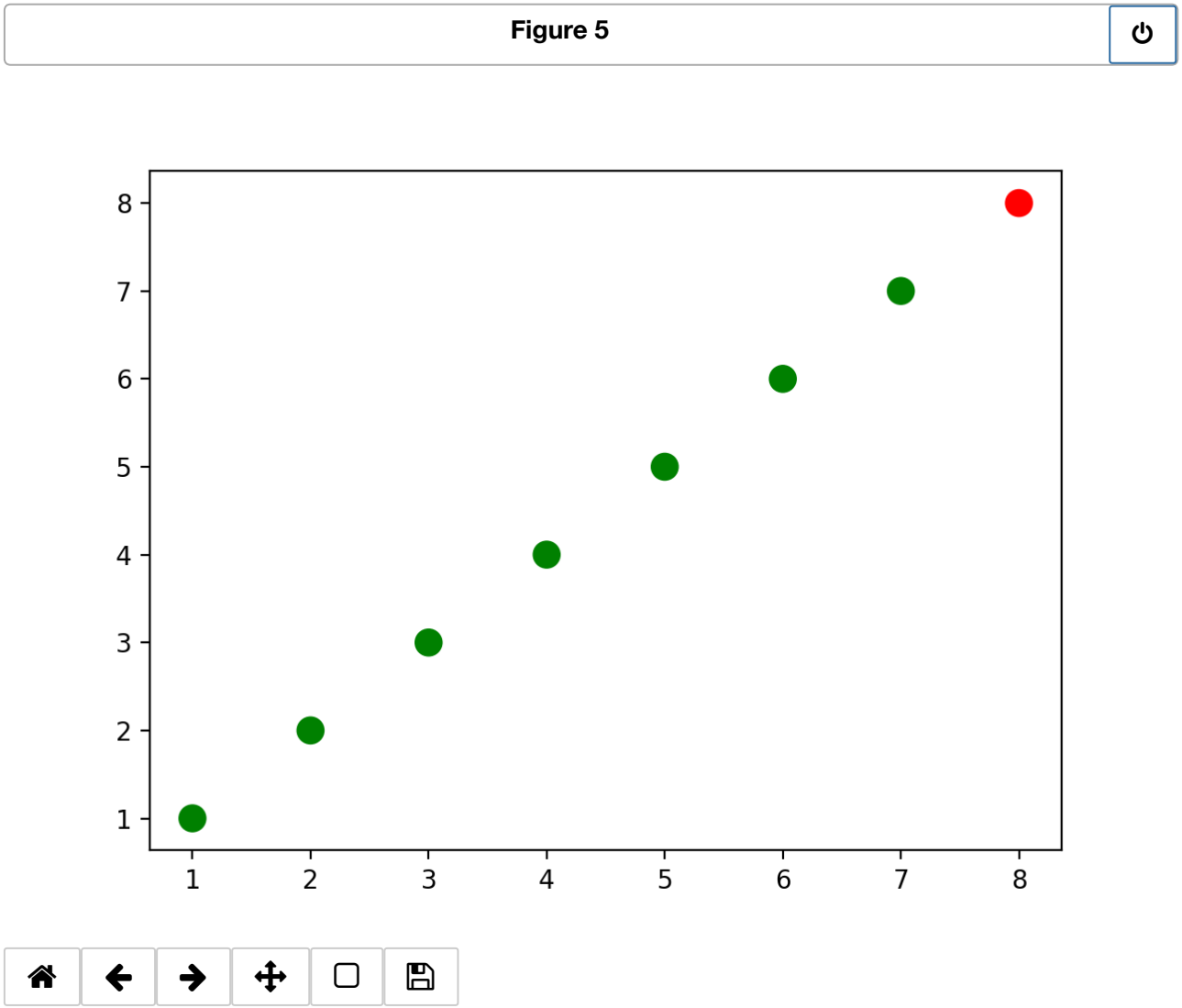
In [37]:
```
import numpy as np

x = np.array([1,2,3,4,5,6,7,8])
y = x

plt.figure()
plt.scatter(x, y) # similar to plt.plot(x, y, '.'), but the underlying child objects in the axes are not Line2D
```

Figure 12



Out[37]: <matplotlib.collections.PathCollection at 0x7f92e88da9b0>

In [12]:
```
import numpy as np

x = np.array([1,2,3,4,5,6,7,8])
y = x

# create a list of colors for each point to have
# ['green', 'green', 'green', 'green', 'green', 'green', 'green', 'red']
colors = ['green']*(len(x)-1)
colors.append('red')

plt.figure()

# plot the point with size 100 and chosen colors
plt.scatter(x, y, s=100, c=colors)
```



Out[12]: <matplotlib.collections.PathCollection at 0x7f92eae9c828>

In [13]:
```
# convert the two lists into a list of pairwise tuples
zip_generator = zip([1,2,3,4,5], [6,7,8,9,10])

print(list(zip_generator))
# the above prints:
# [(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]

zip_generator = zip([1,2,3,4,5], [6,7,8,9,10])
# The single star * unpacks a collection into positional arguments
print(*zip_generator)
# the above prints:
# (1, 6) (2, 7) (3, 8) (4, 9) (5, 10)
```

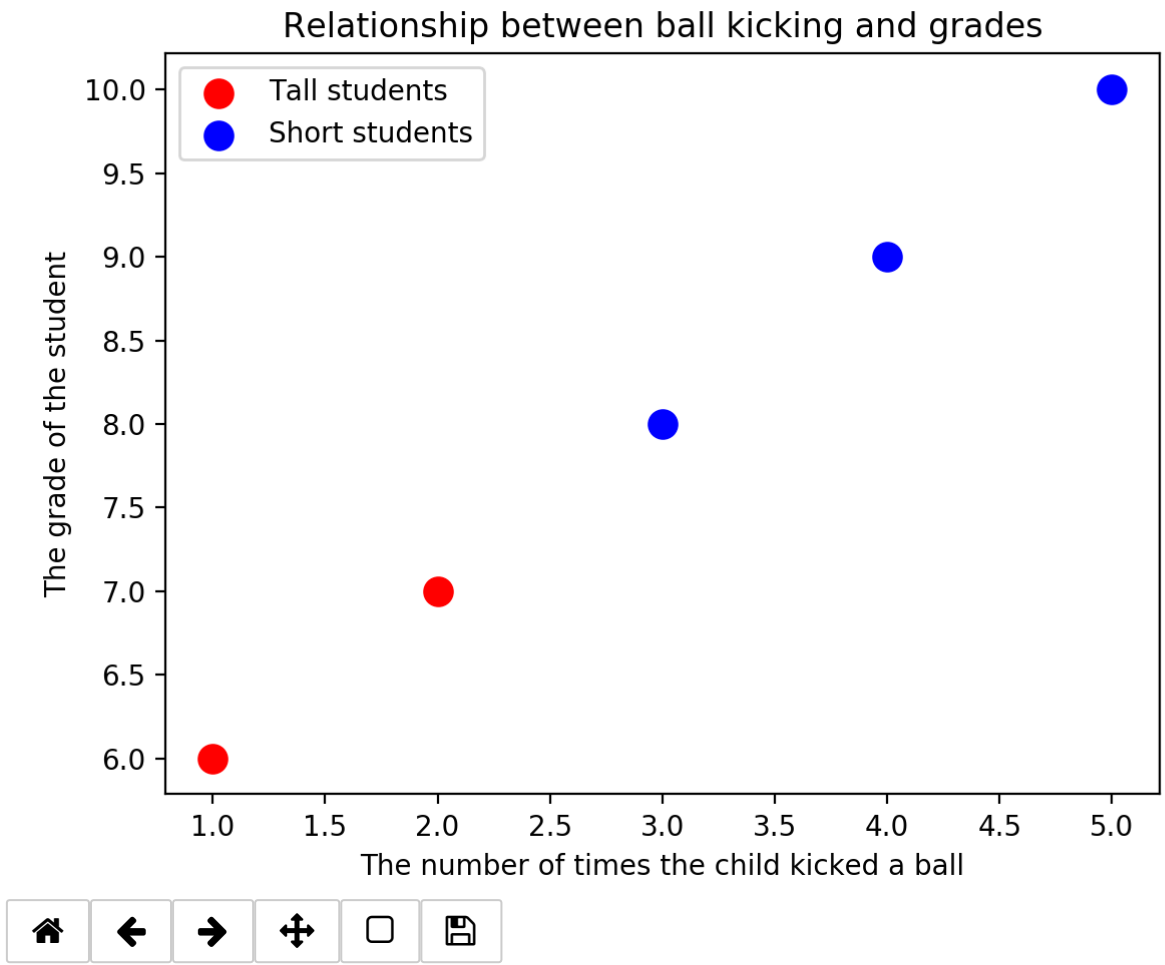
```
[(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]
(1, 6) (2, 7) (3, 8) (4, 9) (5, 10)
```

In [14]:
```
# use zip to convert 5 tuples with 2 elements each to 2 tuples with 5 elements each
print(list(zip((1, 6), (2, 7), (3, 8), (4, 9), (5, 10))))
# the above prints:
# [(1, 2, 3, 4, 5), (6, 7, 8, 9, 10)]


zip_generator = zip([1,2,3,4,5], [6,7,8,9,10])
# let's turn the data back into 2 lists
x, y = zip(*zip_generator) # This is like calling zip((1, 6), (2, 7), (3, 8), (4, 9), (5, 10))
print(x)
print(y)
# the above prints:
# (1, 2, 3, 4, 5)
# (6, 7, 8, 9, 10)
```

```
[(1, 2, 3, 4, 5), (6, 7, 8, 9, 10)]
(1, 2, 3, 4, 5)
(6, 7, 8, 9, 10)
```

In [15]:
```
plt.figure()
# plot a data series 'Tall students' in red using the first two elements of x and y
plt.scatter(x[:2], y[:2], s=100, c='red', label='Tall students')
# plot a second data series 'Short students' in blue using the last three elements of x and y
plt.scatter(x[2:], y[2:], s=100, c='blue', label='Short students')
```

Figure 6



Out[15]: <matplotlib.collections.PathCollection at 0x7f92eae724e0>

In [16]:
```
# add a label to the x axis
plt.xlabel('The number of times the child kicked a ball')
# add a label to the y axis
plt.ylabel('The grade of the student')
# add a title
plt.title('Relationship between ball kicking and grades')
```

Out[16]:
```
<matplotlib.text.Text at 0x7f92eae54e48>
```

In [17]:
```
# add a legend (uses the labels from plt.scatter)
plt.legend()
```

Out[17]:
```
<matplotlib.legend.Legend at 0x7f92eaefce80>
```

In [18]:
```
# add the legend to loc=4 (the lower right hand corner), also gets rid of the frame and adds a title
plt.legend(loc=4, frameon=False, title='Legend')
```

Out[18]:
```
<matplotlib.legend.Legend at 0x7f92eadfd6d8>
```

In [19]:
```
# get children from current axes (the legend is the second to last item in this list)
plt.gca().get_children()
```

Out[19]:
```
[<matplotlib.collections.PathCollection at 0x7f92eae6b908>,
 <matplotlib.collections.PathCollection at 0x7f92eae724e0>,
 <matplotlib.spines.Spine at 0x7f92eaeaf7b8>,
 <matplotlib.spines.Spine at 0x7f92eaeaf9e8>,
 <matplotlib.spines.Spine at 0x7f92eaeafbe0>,
 <matplotlib.spines.Spine at 0x7f92eaeafdd8>,
 <matplotlib.axis.XAxis at 0x7f92eaeaff98>,
 <matplotlib.axis.YAxis at 0x7f92eae39668>,
 <matplotlib.text.Text at 0x7f92eae54e48>,
 <matplotlib.text.Text at 0x7f92eae54eb8>,
 <matplotlib.text.Text at 0x7f92eae54f28>,
 <matplotlib.legend.Legend at 0x7f92eadfd6d8>,
 <matplotlib.patches.Rectangle at 0x7f92eae54f60>]
```

In [20]:
```
# get the legend from the current axes
legend = plt.gca().get_children()[-2]
```

In [21]:
```
# you can use get_children to navigate through the child artists
legend.get_children()[0].get_children()[1].get_children()[0].get_children()
```

Out[21]:
```
[<matplotlib.offsetbox.HPacker at 0x7f92eae043c8>,
 <matplotlib.offsetbox.HPacker at 0x7f92eae04438>]
```

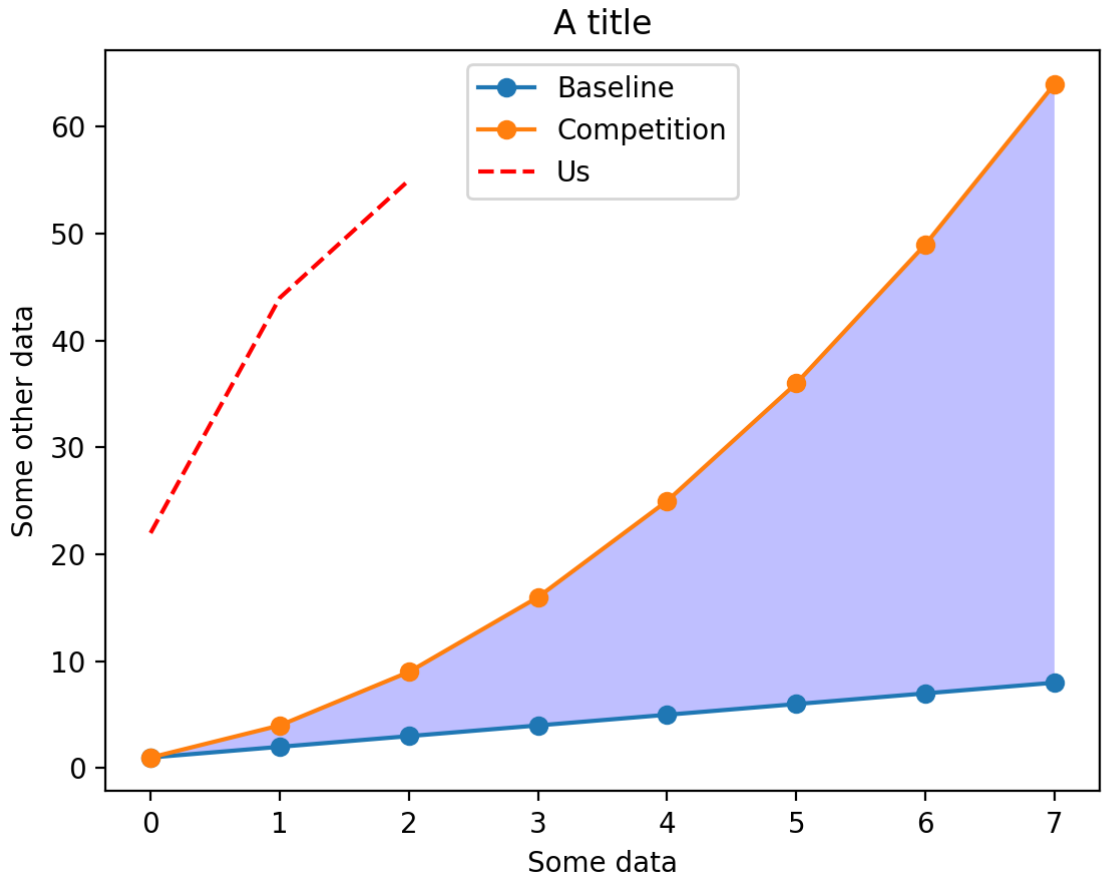
In [22]:
```
# import the artist class from matplotlib
from matplotlib.artist import Artist

def rec_gc(art, depth=0):
    if isinstance(art, Artist):
        # increase the depth for pretty printing
        print("  " * depth + str(art))
        for child in art.get_children():
            rec_gc(child, depth+2)

# Call this function on the legend artist to see what the legend is made up of
rec_gc(plt.legend())
```

```
Legend
    <matplotlib.offsetbox.VPacker object at 0x7f92eae15358>
        <matplotlib.offsetbox.TextArea object at 0x7f92eae15080>
            Text(0,0,'None')
        <matplotlib.offsetbox.HPacker object at 0x7f92eae0e550>
            <matplotlib.offsetbox.VPacker object at 0x7f92eae0e588>
                <matplotlib.offsetbox.HPacker object at 0x7f92eae0ef98>
                    <matplotlib.offsetbox.DrawingArea object at 0x7f92eae0e7f0>
                        <matplotlib.collections.PathCollection object at 0x7f92eae0e9e8>
                    <matplotlib.offsetbox.TextArea object at 0x7f92eae0e5c0>
                        Text(0,0,'Tall students')
                <matplotlib.offsetbox.HPacker object at 0x7f92eae15048>
                    <matplotlib.offsetbox.DrawingArea object at 0x7f92eae0ed30>
                        <matplotlib.collections.PathCollection object at 0x7f92eae0ef28>
                    <matplotlib.offsetbox.TextArea object at 0x7f92eae0ea58>
                        Text(0,0,'Short students')
    FancyBboxPatch(0,0;1x1)
```

Line Plots

In [23]:
```
import numpy as np

linear_data = np.array([1,2,3,4,5,6,7,8])
exponential_data = linear_data**2

plt.figure()
# plot the linear data and the exponential data
plt.plot(linear_data, '-o', exponential_data, '-o')
```

Figure 7



Out[23]:
```
[<matplotlib.lines.Line2D at 0x7f92eade91d0>,
 <matplotlib.lines.Line2D at 0x7f92eade9320>]
```

In [24]:
```
# plot another series with a dashed red line
plt.plot([22,44,55], '--r')
```

Out[24]:
```
[<matplotlib.lines.Line2D at 0x7f92eae15940>]
```

In [25]:
```
plt.xlabel('Some data')
plt.ylabel('Some other data')
plt.title('A title')
# add a legend with legend entries (because we didn't have labels when we plotted the data series)
plt.legend(['Baseline', 'Competition', 'Us'])
```

Out[25]:
```
<matplotlib.legend.Legend at 0x7f92eadef940>
```

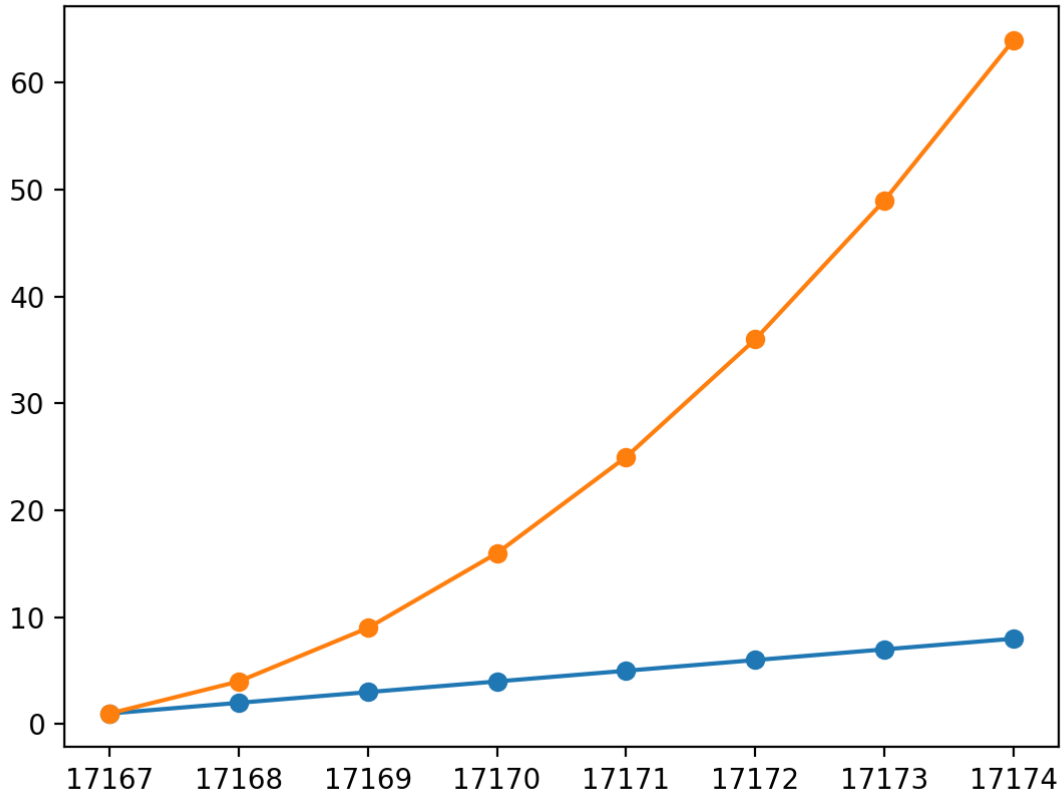
In [26]:
```
# fill the area between the linear data and exponential data
plt.gca().fill_between(range(len(linear_data)),
                       linear_data, exponential_data,
                       facecolor='blue',
                       alpha=0.25)
```

Out[26]:
```
<matplotlib.collections.PolyCollection at 0x7f92eadef828>
```

Let's try working with dates!

In [27]:
```
plt.figure()

observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')

plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')
```

Figure 8



Forward to next view

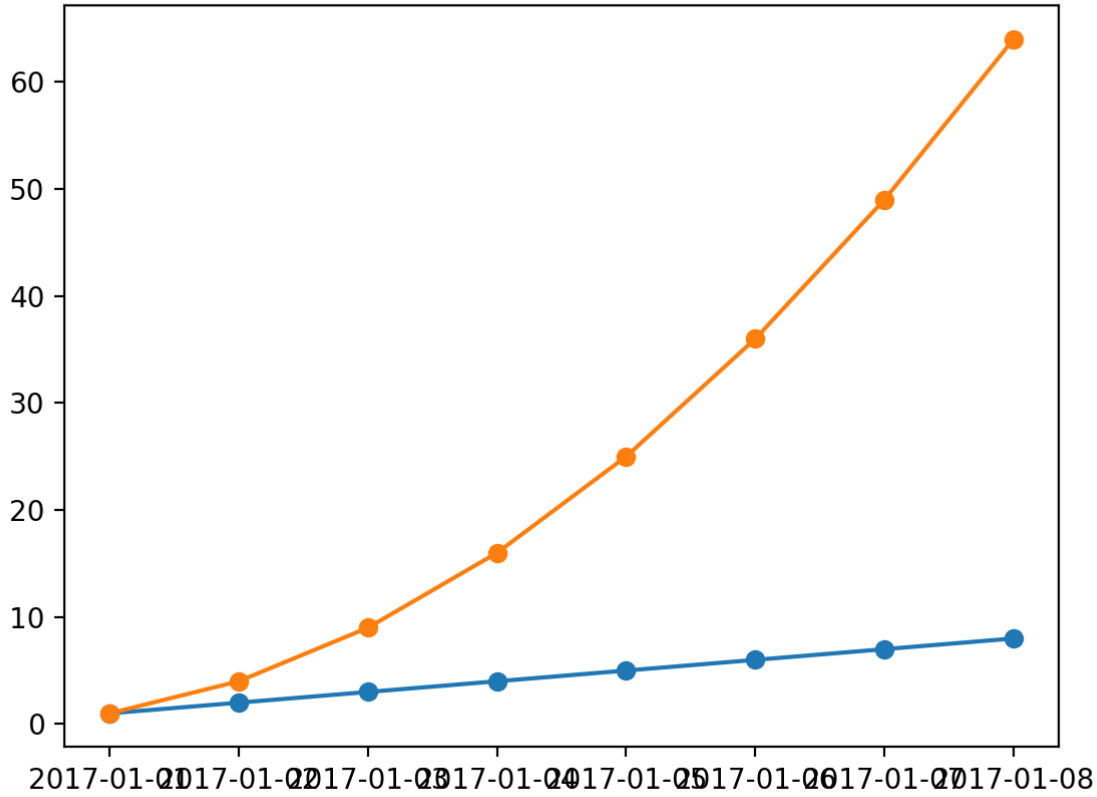
Out[27]:
```
[<matplotlib.lines.Line2D at 0x7f92ead4d198>,
 <matplotlib.lines.Line2D at 0x7f92ead4d320>]
```

Let's try using pandas

In [39]:
```
import pandas as pd

# plt.figure()
# observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')
# observation_dates = map(pd.to_datetime, observation_dates) # trying to plot a map will result in an error
# plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')
```

In [40]:
```
plt.figure()
observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')
observation_dates = list(map(pd.to_datetime, observation_dates)) # convert the map to a list to get rid of the e
plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')
```

Figure 13



Out[40]:
```
[<matplotlib.lines.Line2D at 0x7f92d8d04d68>,
 <matplotlib.lines.Line2D at 0x7f92d8ca9b70>]
```

In [ ]:
```
x = plt.gca().xaxis

# rotate the tick labels for the x axis
for item in x.get_ticklabels():
    item.set_rotation(45)
```

In [ ]:
```
# adjust the subplot so the text doesn't run off the image
plt.subplots_adjust(bottom=0.25)
```

In [ ]:
```
ax = plt.gca()
ax.set_xlabel('Date')
ax.set_ylabel('Units')
ax.set_title('Exponential vs. Linear performance')
```

In [ ]:
```
# you can add mathematical expressions in any text element
ax.set_title("Exponential ($x^2$) vs. Linear ($x$) performance")
```

Bar Charts

In [ ]:
```
plt.figure()
xvals = range(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3)
```

In [ ]:
```
new_xvals = []

# plot another set of bars, adjusting the new xvals to make up for the first set of bars plotted
for item in xvals:
    new_xvals.append(item+0.3)

plt.bar(new_xvals, exponential_data, width = 0.3 ,color='red')
```

In [ ]:
```
from random import randint
linear_err = [randint(0,15) for x in range(len(linear_data))]

# This will plot a new set of bars with errorbars using the list of random error values
plt.bar(xvals, linear_data, width = 0.3, yerr=linear_err)
```

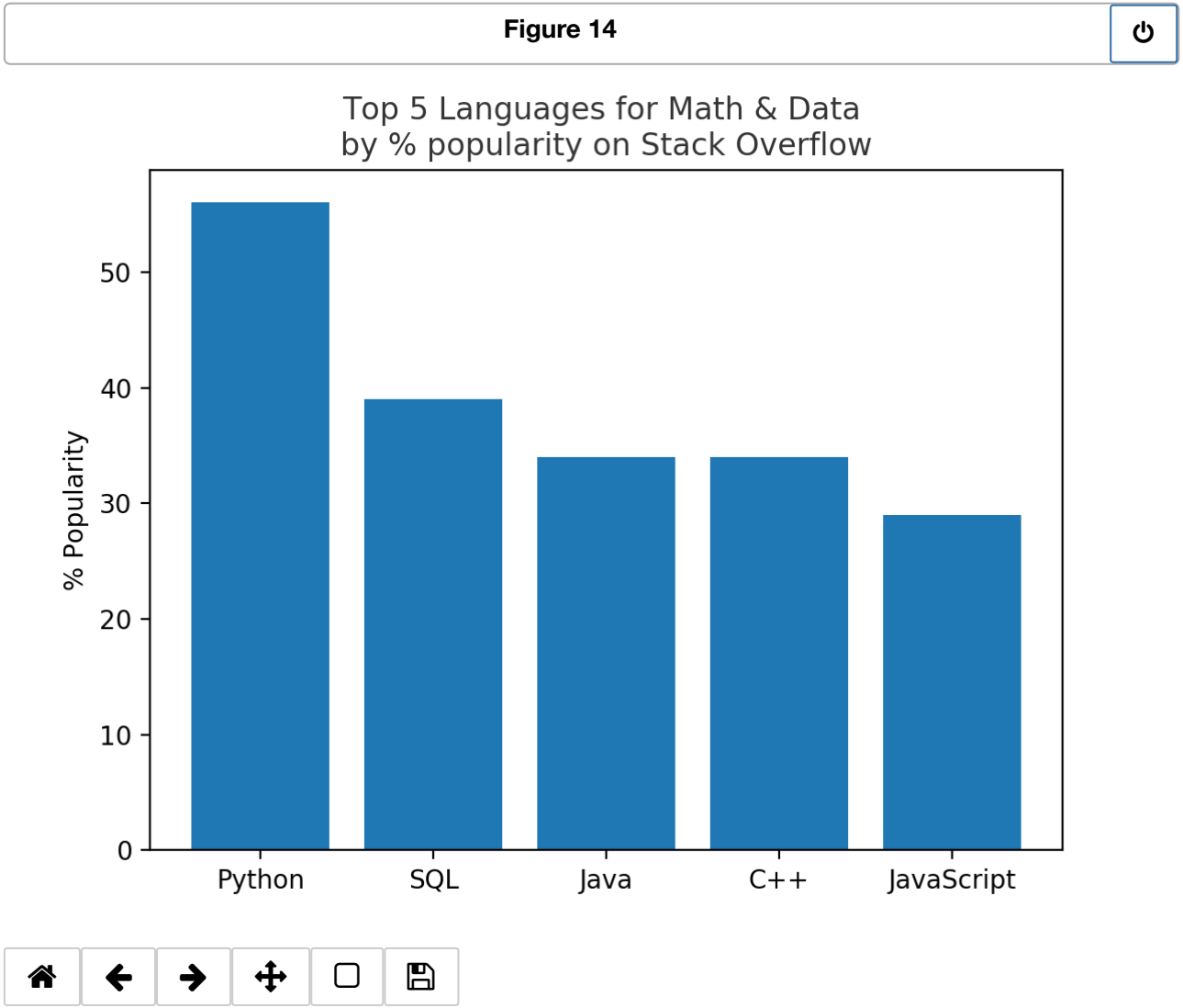
In [ ]:
```
# stacked bar charts are also possible
plt.figure()
xvals = range(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3, color='b')
plt.bar(xvals, exponential_data, width = 0.3, bottom=linear_data, color='r')
```

In [ ]:
```
# or use barh for horizontal bar charts
plt.figure()
xvals = range(len(linear_data))
plt.barh(xvals, linear_data, height = 0.3, color='b')
plt.barh(xvals, exponential_data, height = 0.3, left=linear_data, color='r')
```

Dejunkifying

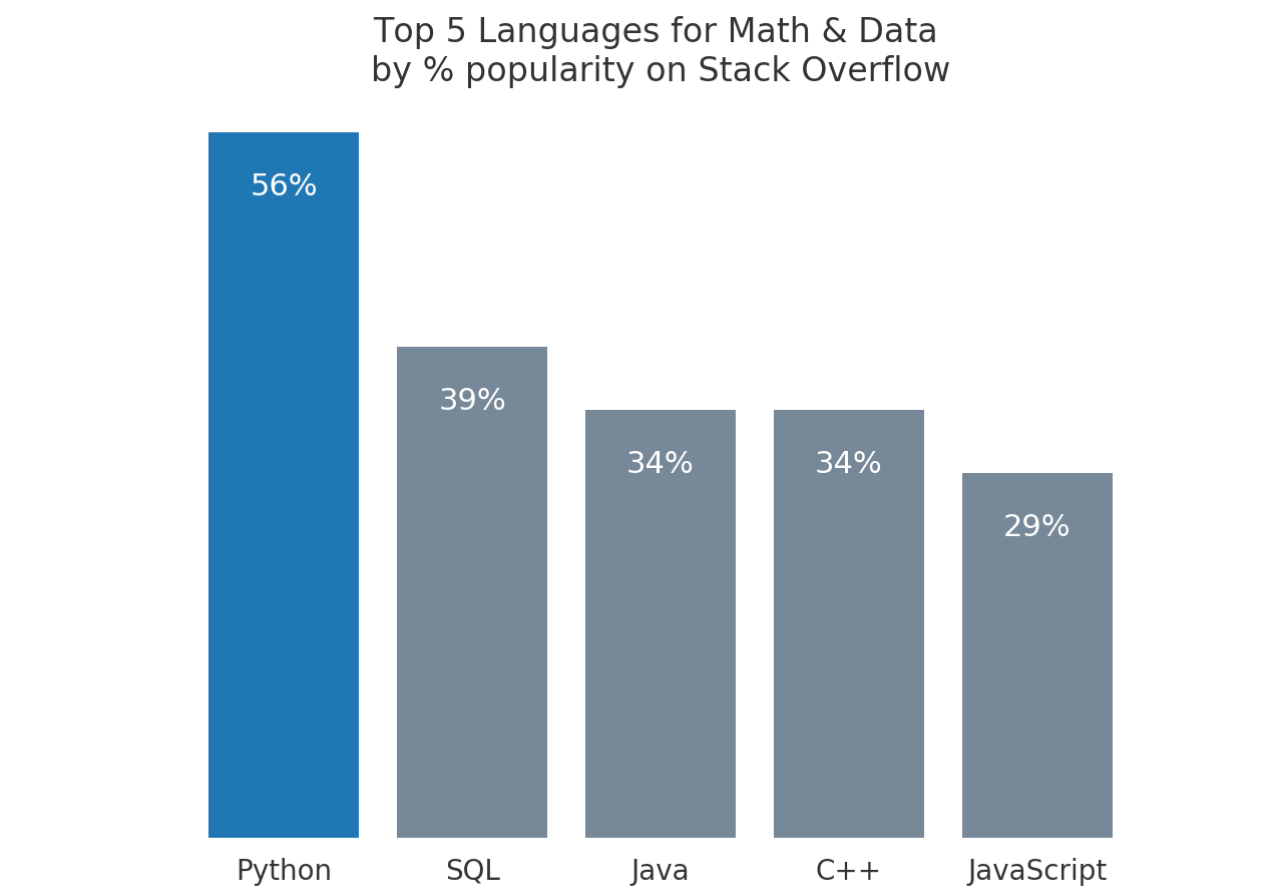
In [41]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()

languages =['Python', 'SQL', 'Java', 'C++', 'JavaScript']
pos = np.arange(len(languages))
popularity = [56, 39, 34, 34, 29]

plt.bar(pos, popularity, align='center')
plt.xticks(pos, languages)
plt.ylabel('% Popularity')
plt.title('Top 5 Languages for Math & Data \nby % popularity on Stack Overflow', alpha=0.8)

plt.show()
```

Figure 14


Top 5 Languages for Math & Data
by % popularity on Stack Overflow
% Popularity
0
10
20
30
40
50
Python
SQL
Java
C++
JavaScript

In [42]:
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()

languages =['Python', 'SQL', 'Java', 'C++', 'JavaScript']
pos = np.arange(len(languages))
popularity = [56, 39, 34, 34, 29]

# change the bar color to be less bright blue
bars = plt.bar(pos, popularity, align='center', linewidth=0, color='lightslategrey')
# make one bar, the python bar, a contrasting color
bars[0].set_color('#1F77B4')

# soften all labels by turning grey
plt.xticks(pos, languages, alpha=0.8)
# remove the Y label since bars are directly labeled
#plt.ylabel('% Popularity', alpha=0.8)
plt.title('Top 5 Languages for Math & Data \nby % popularity on Stack Overflow', alpha=0.8)

# remove all the ticks (both axes), and tick labels on the Y axis
plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off', labelbottom='on')

# remove the frame of the chart
for spine in plt.gca().spines.values():
    spine.set_visible(False)

# direct label each bar with Y axis values
for bar in bars:
    plt.gca().text(bar.get_x() + bar.get_width()/2, bar.get_height() - 5, str(int(bar.get_height())) + '%',
                   ha='center', color='w', fontsize=11)
plt.show()
```

Figure 15



x= y=4.51868

The end